CENG 218 - Lab Work 4 – Week 4

Introduction

Linked Lists in C++ Library (Standard Template Library (STL))

Forward list

Forward lists are sequence containers that allow **constant** time insert and **erase** operations **anywhere** within the sequence. Forward lists are implemented as **singly-linked** lists; Singly linked lists can store each of the elements they contain in **different and unrelated storage locations**. The ordering is kept by the association to each element of a link to the next element in the sequence. (Source: https://cplusplus.com/reference/forward_list/forward_list/)

When to Consider std::forward_list in you applications

- **Insertion, Removal, and Moving Elements at Any Position:** Forward lists excel in operations where you need to frequently add, remove, or move elements anywhere within the container, especially at the front.
- **Memory Efficiency:** If you only need to iterate through elements in one direction (from beginning to end) and memory usage is a major concern, a forward list often uses less overhead than a std::list.
- **Simple Iteration:** For straightforward sequential access to elements, a forward list is a suitable choice.
- **Algorithms Requiring Single-Pass:** Forward lists work particularly well with algorithms that need to traverse the sequence once, like some sorting algorithms.

Key Considerations

- **No Random Access:** Unlike std::vector or std::list, you cannot directly access an element by its index in a forward list. You must iterate sequentially.
- **Only Forward Traversal:** Forward lists, being singly linked lists, only allow you to move from the beginning towards the end. You cannot iterate backward.

Common Scenarios

- **Implementing LRU Caches:** Forward lists can be effective in creating in-memory caches where items are often added to the front with older items being discarded.
- **Event Handling Systems:** Where events need to be processed in the order they arrive.
- **Certain Graph Representations:** Representing graphs where you primarily traverse edges in one direction may benefit from the efficiency of a forward list to store those edges.

Project:

We are planning to start a Car Rental Company called **EliteDrive. We need a simple software that will handle all renatal operations.**

Design a C++ program for a **EliteDrive** company that utilizes a **std::forward_list** to manage a list of available cars and a set of customer records. The program should provide users with a menu-driven interface allowing them to:

1. Display the current inventory of available cars, including details such as car type, brand, model, type, year, and **rental status**.
2. Rent a car by specifying the type, brand and model. If the selected car is available, mark it as rented and associate the rental with the customer.
3. Return a rented car by specifying the brand and model. If the selected car is currently rented, mark it as returned and update the customer's rental history.
4. Display customer information, including their rental history, name, contact details. (if you you may further additional other relevant information)
5. Register new customers, allowing them to provide personal information such as name, contact details.
6. View customer accounts, booking history, and make new reservations.
7. Exit the application.

EliteDrive will offer a variety of car types to cater to different customer needs and preferences. Following car types considered in our rental fleet:

1. **Economy Cars:**
   - Examples: Toyota Yaris, Ford Fiesta
   - Features: Compact size, fuel efficiency, budget-friendly
2. **Compact Cars:**
   - Examples: Honda Civic, Chevrolet Cruze
   - Features: Slightly larger than economy cars, good fuel efficiency, comfortable for small groups
3. **Midsize Cars:**
   - Examples: Toyota Camry, Ford Fusion
   - Features: More spacious than compact cars, suitable for families, good balance of size and fuel efficiency
4. **Full-Size Cars:**
   - Examples: Chevrolet Impala, Nissan Maxima
   - Features: Larger and more comfortable, suitable for longer trips or groups with more luggage
5. **SUVs (Sports Utility Vehicles):**
   - Examples: Ford Explorer, Toyota RAV4
   - Features: Versatile for various terrains, spacious, and often equipped with advanced safety features
6. **Crossovers:**
   - Examples: Honda CR-V, Nissan Rogue
   - Features: Combines features of SUVs and sedans, providing a smooth ride and ample cargo space
7. **Luxury Cars:**
   - Examples: BMW 7 Series, Mercedes-Benz S-Class
   - Features: High-end amenities, advanced technology, and superior comfort
8. **Convertibles:**
   - Examples: Mazda MX-5, Ford Mustang Convertible
   - Features: Open-top driving experience, suitable for leisure or special occasions
9. **Vans/Minivans:**
   - Examples: Chrysler Pacifica, Honda Odyssey
   - Features: Ideal for family trips or transporting larger groups, often equipped with sliding doors
10. **Trucks:**
    - Examples: Ford F-150, Chevrolet Silverado
    - Features: Utility vehicles suitable for transporting goods or for customers requiring towing capabilities
11. **Electric or Hybrid Cars:**
    - Examples: Tesla Model S (Electric), Toyota Prius (Hybrid)
    - Features: Environmentally friendly, fuel-efficient, and often equipped with modern technology

For each car type EliteDrive has 5 cars.

Your application will utilize cars.txt file rather than using literals.

```
CarType,Brand,Model,Year
Economy,Toyota,Yaris,2022
.
.
.

Electric,Tesla,Model S,2022
Hybrid,Toyota,Prius,2023
```